Paleti Krishnasai
CED18I039

Operating Systems - Assignments 7 & 8 - SYNCHRONIZATION

1. Simulate the Producer Consumer code discussed in the class.

Code:

```
// simulate producer consumer code discussed in the class

// 1 producer 3 consumers using semaphores

//Assumption: Buffer Size = 1
//We manually define the 3 consumers the consume exactly one-third of the
items produced by the Producer for ease of Understanding.

#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define SHARED 1

void * Producer();
void *Consumer();

sem_t empty, full, sm;
int data;

int main()
{
   pthread_t ptid, ctid1, ctid2, ctid3;
   int x = 1, y = 2, z = 3;
   int *a = &x;
   int *b = &y;
   int *c = &z;
   sem_init(&empty, SHARED, 1);
   sem_init(&full, SHARED, 0);
   sem_init(&sm, SHARED, 1);
```

```
   pthread_create(&ptid, NULL, Producer, NULL);
   pthread_create(&ctid1, NULL, Consumer, (void *)a);
   pthread_create(&ctid2, NULL, Consumer, (void *)b);
   pthread_create(&ctid3, NULL, Consumer, (void *)c);
   pthread_join(ptid, NULL);
   pthread_join(ctid1, NULL);
   pthread_join(ctid2, NULL);
   pthread_join(ctid3, NULL);
   printf("\nMain done\n");
}

void *Producer()
{
   int produced;
   printf("\nProducer created");
   printf("\nProducer id is %ld\n", pthread_self());
   for (produced = 0; produced < 30; produced++)
   {
       sem_wait(&empty);
       sem_wait(&sm);
       data = produced;
       sem_post(&sm);
       sem_post(&full);
       printf("\nProducer: %d", data);
   }
}

void *Consumer(void *no)
{
   int consumed, total = 0;
   int *thread = (int *)no;
   printf("\nConsumer created, Thread number: %d\n", *thread);
   printf("\nConsumer id is %ld\n", pthread_self());
   for (consumed = 0; consumed < 10; consumed++)
   {
       sem_wait(&full);
       sem_wait(&sm);
       total = total + data;
       printf("\nThread: %d, Consumed: %d", *thread, data);
       sem_post(&sm);
```

```
        sem_post(&empty);
    }
    printf("\nThe total of 10 iterations for thread %d is %d\n", *thread,
total);
}
```

Output:

```
paleti@paletil:~/OS_LAB/Synchronization$ gcc pc.c -lpthread -lrt
paleti@paletil:~/OS_LAB/Synchronization$ ./a.out

Producer created
Producer id is 139886170167040

Producer: 0
Consumer created, Thread number: 3

Consumer id is 139886011086592

Thread: 3, Consumed: 0
Consumer created, Thread number: 2

Consumer id is 139886153381632

Consumer created, Thread number: 1

Consumer id is 139886161774336

Thread: 3, Consumed: 1
Producer: 1
Producer: 2
Thread: 2, Consumed: 2
Producer: 3
Thread: 1, Consumed: 3
Producer: 4
Thread: 3, Consumed: 4
Producer: 5
Thread: 2, Consumed: 5
Producer: 6
Thread: 1, Consumed: 6
Producer: 7
Thread: 3, Consumed: 7
Producer: 8
Thread: 2, Consumed: 8
Producer: 9
Thread: 1, Consumed: 9
Producer: 10
Thread: 3, Consumed: 10
Producer: 11
Thread: 2, Consumed: 11
Producer: 12
Thread: 1, Consumed: 12
Producer: 13
Thread: 3, Consumed: 13
Producer: 14
Thread: 2, Consumed: 14
Producer: 15
Thread: 1, Consumed: 15
Producer: 16
Thread: 3, Consumed: 16
Producer: 17
Thread: 2, Consumed: 17
Producer: 18
Thread: 1, Consumed: 18
Producer: 19
Thread: 3, Consumed: 19
Producer: 20
Thread: 2, Consumed: 20
Producer: 21
Thread: 1, Consumed: 21
```

Paleti Krishnasai
CED18I039

```
Thread: 1, Consumed: 21
Producer: 22
Thread: 3, Consumed: 22
Producer: 23
Thread: 2, Consumed: 23
Producer: 24
Thread: 1, Consumed: 24
Producer: 25
Thread: 3, Consumed: 25
The total of 10 iterations for thread 3 is 117

Producer: 26
Thread: 2, Consumed: 26
Producer: 27
Thread: 1, Consumed: 27
Producer: 28
Thread: 2, Consumed: 28
The total of 10 iterations for thread 2 is 154

Thread: 1, Consumed: 29
The total of 10 iterations for thread 1 is 164

Producer: 29
Main done
paleti@paletil:~/OS_LAB/Synchronization$
```

Paleti Krishnasai
CED18I039

2. Extend the producer consumer simulation in Q1 to sync access of critical data using Peterson's Algorithm.

Reference

Both producer and consumer function uses lock and unlock function to lock process.Lock function sets turn to the other process value and sets its process flag value to true similar to the code discussed in the class.Unlock() sets flag to false.

Code:

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define SHARED 1

int data;
int flag[2];
int turn;
sem_t empty,full;

void lock_in()
{
   flag[0] = 0;
   flag[1] = 0;
   turn = 0;

}

void lock(int index)
{
   flag[index] = 1;
   turn = 1-index;
   while(flag[1-index]==1 && turn==1-index);
}

void unlock(int index)
{
   flag[index]=0;
}
```

Paleti Krishnasai
CED18I039

```
void *producer()
{
   int p_no;
   printf("\nNew Producer id : %ld\n",pthread_self());
   for(p_no=0;p_no<5;p_no++)
   {
       sem_wait(&empty);
       lock(0);
       //CS starts
       data = p_no;
       //CS ends
       unlock(0);
       sem_post(&full);
       printf("\nProduced data : %d\n",data);
   }
}

void *consumer()
{
   int c_no,total=0;
   printf("\nNew consumer id : %ld\n",pthread_self());
   for(c_no=0;c_no<5;c_no++)
   {
       sem_wait(&full);
       lock(1);
       //CS starts
       total = total + data;
       //CS ends
       unlock(1);
       sem_post(&empty);
       printf("\nconsumed data : %d\n",data);
   }
   printf("\nTotal consumed data %d\n",total);
}

int main()
{
   pthread_t p1,c1;
   lock_in();
```

```
    sem_init(&empty,SHARED,1);
    sem_init(&full,SHARED,0);
    pthread_create(&p1,NULL,producer,NULL);
    pthread_create(&c1,NULL,consumer,NULL);
    pthread_join(p1,NULL);
    pthread_join(c1,NULL);
    printf("\nSTOPPED\n");
    return 0;
}
```

Output:

```
paleti@paletil:~/OS_LAB/Synchronization$ ./a.out

New consumer id : 139918594094848

New Producer id : 139918602487552

Produced data : 0

consumed data : 0

consumed data : 1

Produced data : 1

Produced data : 2

Produced data : 3

consumed data : 2

consumed data : 3

Produced data : 4

consumed data : 4

Total consumed data 10

STOPPED
paleti@paletil:~/OS_LAB/Synchronization$
```

Paleti Krishnasai
CED18I039

3. Dictionary Problem: Let the producer set up a dictionary of at least 20 words with three attributes (Word, Primary meaning, Secondary meaning) and let the consumer search for the word and retrieve its respective primary and secondary meaning.
Note: This can be implemented using either Mutex locks or Peterson's algorithm.

Non-Mandatory (Extra credits):

4. Extend Q3 to avoid duplication of dictionary entries and implement an efficient binary search on the consumer side in a multithreaded fashion.

Get words and build a dictionary in the producer and pass the word alone to the consumer. The consumer searchd the word in the global dict.
A new for loop has been created so that when a new word is taken from the file ,it will be checked whether its already included in the array or not.

Code:

```
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
#include<string.h>
#include <stdlib.h>
#define SHARED 1

//pete algo vars
int flag[2];
int turn;
//sem vars for buffer
sem_t empty,full;

struct node
{
   char word_s[1000];
   int index;
   int half;
```

```
};

//dict vars
char word[100][1000];
char primary[100][1000];
char secondary[100][1000];
int size;

void lock_ini()
{
   flag[0]=0;
   flag[1]=0;
   turn=0;
}


void swap_word(int i,int j)
{
   char temp[1000]="";
   strcpy(temp,word[i]);
   strcpy(word[i],word[j]);
   strcpy(word[j],temp);
}
void swap_p(int i,int j)
{
   char temp[1000]="";
   strcpy(temp,primary[i]);
   strcpy(primary[i],primary[j]);
   strcpy(primary[j],temp);
}
void swap_s(int i,int j)
{
   char temp[1000]="";
   strcpy(temp,secondary[i]);
   strcpy(secondary[i],secondary[j]);
   strcpy(secondary[j],temp);
}

void bubblesort(int ch,int start ,int end)
{
```

```
int i,j;
for(i=start;i<end-1;i++)
{
for(j=start;j<end-(i-start)-1;j++)
{

int
choice=(ch==0)?(strcmp(word[j],word[j+1])<0):(strcmp(word[j],word[j+1])>0)
;
if(choice==1)
{
swap_word(j,j+1);
swap_p(j,j+1);
swap_s(j,j+1);

}
}
}
}

int bs(int start,int end,char key[1000])
{

while(start<=end)
{
int mid=start+(end-start)/2;
if(strcmp(word[mid],key)==0)
return mid;
else if(strcmp(word[mid],key)<0)
start=mid+1;
else
end=mid-1;
}
return -1;
}

void lock(int index)
{
   flag[index]=1;
   turn=1-index;
```

```
   while(flag[1-index]==1 && turn==1-index);
}
void unlock(int index)
{
   flag[index]=0;
}

void create_dict()
{

   FILE *fp=fopen("dict.txt","r");
   char c=fgetc(fp);
   //while(c!=feof(fp))
   for(int i=0;i<size;i++)
   {
      int index=0;
     while(c!=';')
       {
           word[i][index++]=c;
           c=fgetc(fp);

       }
       index=0;
       c=fgetc(fp);
       while(c!=';')
       {
           primary[i][index++]=c;
           c=fgetc(fp);

       }
       index=0;
       c=getc(fp);
       while(c!='$')
       {
           secondary[i][index++]=c;
           c=fgetc(fp);

       }
       for(int j=0;j<2;j++)
           c=fgetc(fp);
```

```
    for(int j=0;j<i;j++)
       {
           if(strcmp(word[j],word[i])==0)
           {
               memset(word[i],0,sizeof(word[i]));
               memset(primary[i],0,sizeof(primary[i]));
               memset(secondary[i],0,sizeof(secondary[i]));
               size-=1;
               i-=1;

           }
       }

   }
}
void findsize()
{
   FILE *fp=fopen("dict.txt","r");
   char c;
   for (c = getc(fp); c != EOF; c = getc(fp))
       if (c == '\n') // Increment count if this character is newline
           size+=1;
   fclose(fp);
}
void *producer()
{
findsize();
sem_wait(&empty);
//entry-section
      lock(0);
          //cs start
                   //assume data to the new producer no
                  create_dict();
          //cs end
      unlock(0);
//exit section
sem_post(&full);

pthread_exit(NULL);
```

```
}

void* process(void *arg)
{
  struct node *temp=(struct node*)arg;

  if(temp->half==1)
  {
      bubblesort(1,0,size/2);
      temp->index=bs(0,size/2-1,temp->word_s);

  }
  else
  {
      bubblesort(1,size/2,size);
       temp->index=bs(size/2,size-1,temp->word_s);

   }

  pthread_exit(NULL);
}

void search()
{
   pthread_t pth[2];

  struct node* keys=(struct node*)malloc(2*sizeof(struct node));
   printf("Enter the word to be searched in 1st half:\n");
   scanf("%s",keys[0].word_s);

   printf("\nEnter the word to be searched in 2nd half:\n");
   scanf("%s",keys[1].word_s);
   keys[0].half=1;
   keys[1].half=2;

   pthread_create(&pth[0],NULL,process,&keys[0]);
   pthread_create(&pth[1],NULL,process,&keys[1]);

   for(int i=0;i<2;i++)
   {
```

```
  pthread_join(pth[i],NULL);
   }
    if(keys[0].index!=-1)
      {
           printf("\nWord Found By the Consumer in 1st
Half:\n\nWord:%s\nPrimary Meaning:%s\nSecondary
Meaning:%s\n\n",word[keys[0].index],primary[keys[0].index],secondary[keys[
0].index]);
      }
      else
        printf("Word Not found By the Consumer as its not in the
Dictionary(1st half)!!!\n\n");

   if(keys[1].index!=-1)
      {
           printf("\nWord Found By the Consumer in 2nd
Half:\n\nWord:%s\nPrimary Meaning:%s\nSecondary
Meaning:%s\n\n",word[keys[1].index],primary[keys[1].index],secondary[keys[
1].index]);
      }
      else
        printf("Word Not found By the Consumer as its not in the
Dictionary(2nd half)!!!\n\n");
}
void *consumer()
{

sem_wait(&full);
//entry-section
      lock(1);
             //cs start
                    search();
            //cs end
      unlock(1);
//exit section
sem_post(&empty);

pthread_exit(NULL);
}
```

Paleti Krishnasai
CED18I039

```
//empty to check whether buffer is empty;default 1
//full to check whether buffer is full;default 0
//lock is mutext lock;default 1

int main()
{

   pthread_t p1,c1;
   lock_ini();
   sem_init(&empty,SHARED,1);
   sem_init(&full,SHARED,0);

   pthread_create(&p1,NULL,producer,NULL);
   pthread_create(&c1,NULL,consumer,NULL);

   pthread_join(p1,NULL);
   pthread_join(c1,NULL);

   printf("\nStopped\n");

}
```

Paleti Krishnasai
CED18I039

Output:

Synchronization > dict.txt

```
evade;Escape or avoid through guile or trickery;Avoid someone $
dictionary;a book or electronic resource that lists the words of a language;a reference book on a particular subject$
crazy;mad,especially as manifested in wild or aggressive behaviour;extremely enthusiastic$
insane;in a state of mind which prevents normal perception;outrageous$
horror;an intense feeling of fear, shock, or disgust;a bad or mischievous person, especially a child$
sweet;having the pleasant taste characteristic of sugar or honey; not salt, sour, or bitter;delightful$
disaster;a sudden accident or a natural catastrophe that causes great damage or loss of life.;an event or fact that has unfortunate consequences.$
sense;Perceive by a sense or senses;detect$
leave;Go away from;Allow or cause to remain$
vague;Of uncertain, indefinite, or unclear character or meaning;Thinking or communicating in an unfocused or imprecise way$
happy;Feeling or showing pleasure or contentment;Fortunate and convenient$
strike;Hit forcibly and deliberately with one's hand or a weapon or other implement;Cur suddenly and have harmful or damaging effects on$
death;the action or fact of dying or being killed;the end of the life of a person or organism.$
life;the existence of an individual human being or animal;living$
teenage;denoting a person between 13 and 19 years old;relating to or characteristic of teenagers$
baby;a very young child;a lover or spouse$
garner;gather;collect$
farm;an area of land and its buildings, used for growing crops and rearing animals;send out or subcontract work to others.$
adrenaline;hormone;increases rates of blood circulation$
buddy;a close friend;a working companion with whom close cooperation is required$
```

```
paleti@paletil:~/OS_LAB/Synchronization$ ./a.out
Enter the word to be searched in 1st half:
crazy

Enter the word to be searched in 2nd half:
death

Word Found By the Consumer in 1st Half:

Word:crazy
Primary Meaning:mad,especially as manifested in wild or aggressive behaviour
Secondary Meaning:extremely enthusiastic


Word Found By the Consumer in 2nd Half:

Word:death
Primary Meaning:the action or fact of dying or being killed
Secondary Meaning:the end of the life of a person or organism.
```

Paleti Krishnasai
CED18I039

ASSIGNMENT 8

(A) Implement the Dining Philosophers and Reader Writer Problem of Synchronization (test drive the codes discussed in the class).

Dining Philosopher Problem :

Code:

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N

int state[N];
int phil[N] = {0, 1, 2, 3, 4};

sem_t mutex;
sem_t S[N];

void test(int phnum)
{
   if (state[phnum] == HUNGRY && state[LEFT] != EATING && state[RIGHT] !=
EATING)
   {
       // state that eating
       state[phnum] = EATING;

       sleep(2);

       printf("Philosopher %d takes fork %d and %d\n",
              phnum + 1, LEFT + 1, phnum + 1);
```

```
        printf("Philosopher %d is Eating\n", phnum + 1);

        // sem_post(&S[phnum]) has no effect
        // during takefork
        // used to wake up hungry philosophers
        // during putfork
        sem_post(&S[phnum]);
    }
}

// take up chopsticks
void take_fork(int phnum)
{

    sem_wait(&mutex);

    // state that hungry
    state[phnum] = HUNGRY;

    printf("Philosopher %d is Hungry\n", phnum + 1);

    // eat if neighbours are not eating
    test(phnum);

    sem_post(&mutex);

    // if unable to eat wait to be signalled
    sem_wait(&S[phnum]);

    sleep(1);
}

// put down chopsticks
void put_fork(int phnum)
{

    sem_wait(&mutex);

    // state that thinking
```

```
    state[phnum] = THINKING;

    printf("Philosopher %d putting fork %d and %d down\n",
           phnum + 1, LEFT + 1, phnum + 1);
    printf("Philosopher %d is thinking\n", phnum + 1);

    test(LEFT);
    test(RIGHT);

    sem_post(&mutex);
}

void *philospher(void *num)
{

    while (1)
    {

        int *i = num;

        sleep(1);

        take_fork(*i);

        sleep(0);

        put_fork(*i);
    }
}

int main()
{

    int i;
    pthread_t thread_id[N];

    // initialize the semaphores
    sem_init(&mutex, 0, 1);

    for (i = 0; i < N; i++)
```

```
        sem_init(&S[i], 0, 0);

    for (i = 0; i < N; i++)
    {

        // create philosopher processes
        pthread_create(&thread_id[i], NULL,
                       philospher, &phil[i]);

        printf("Philosopher %d is thinking\n", i + 1);
    }

    for (i = 0; i < N; i++)

        pthread_join(thread_id[i], NULL);
}
```

Paleti Krishnasai
CED18I039

Output :

```
paleti@paletil:~/OS_LAB/Synchronization$ ./a.out
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 is Hungry
Philosopher 2 is Hungry
Philosopher 3 is Hungry
Philosopher 4 is Hungry
Philosopher 5 is Hungry
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 5 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
```

Paleti Krishnasai
CED18I039

Readers-Writers Problem :

Code :

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

#define READ_COUNT 5
#define WRITE_COUNT 7

sem_t writer;
pthread_mutex_t mutex;
int count = 1, reader = 0;

void *writer_f(void *wno)
{
   sem_wait(&writer);
   count = count * 2;
   printf("[W]Writer %d changed the content to %d\n\033[0m", (*((int
*)wno)), count);
   sem_post(&writer);
}

void *reader_f(void *rno)
{
   pthread_mutex_lock(&mutex);
   reader += 1;
   if (reader == 1)
   {
       sem_wait(&writer); // Block the writer if its the first reader
   }

   pthread_mutex_unlock(&mutex);
   printf("[R]Reader %d Read the content as %d\n\033[0m", *((int *)rno),
count);

   pthread_mutex_lock(&mutex);
   reader -= 1;
   if (reader == 0)
```

```
    {
        sem_post(&writer); // Wake up the writer if its the last reader
    }
    pthread_mutex_unlock(&mutex);
}

int main()
{
    printf("\n");
    pthread_t read[READ_COUNT], write[WRITE_COUNT];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&writer, 0, 1);

    int readers[READ_COUNT];
    for (int i = 0; i < READ_COUNT; i++)
        readers[i] = i + 1;

    int writers[WRITE_COUNT];
    for (int i = 0; i < WRITE_COUNT; i++)
        writers[i] = i + 1;

    for (int i = 0; i < WRITE_COUNT; i++)
        pthread_create(&write[i], NULL, (void *)writer_f, (void
*)&writers[i]);

    for (int i = 0; i < READ_COUNT; i++)
        pthread_create(&read[i], NULL, (void *)reader_f, (void
*)&readers[i]);

    for (int i = 0; i < READ_COUNT; i++)
        pthread_join(read[i], NULL);

    for (int i = 0; i < WRITE_COUNT; i++)
        pthread_join(write[i], NULL);

    pthread_mutex_destroy(&mutex);
    sem_destroy(&writer);

    printf("\n");
    return 0;
```

```
}
```

Output :

```
paleti@paletil:~/OS_LAB/Synchronization$ gcc readerwriter.c -lpthread -lrt
paleti@paletil:~/OS_LAB/Synchronization$ ./a.out

[W]Writer 1 changed the content to 2
[W]Writer 2 changed the content to 4
[W]Writer 3 changed the content to 8
[W]Writer 4 changed the content to 16
[W]Writer 5 changed the content to 32
[W]Writer 6 changed the content to 64
[W]Writer 7 changed the content to 128
[R]Reader 1 Read the content as 128
[R]Reader 2 Read the content as 128
[R]Reader 3 Read the content as 128
[R]Reader 4 Read the content as 128
[R]Reader 5 Read the content as 128

paleti@paletil:~/OS_LAB/Synchronization$
```

Paleti Krishnasai
CED18I039

(B) Choose any 2 of the following problems whose details are available in the Downy Book on Semaphores (attached) and implement semaphores based solutions to the same.

(1)Santa Claus Problem
Code:

```c
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <unistd.h>
#define SHARED 1

int elvesCount = 0;
int reindeerCount = 0;
sem_t santa_s, reindeer_s, elf_s, mutex;

void prepare()
{
   printf("[* *] Sleigh is being Prepared!!!\n");
}

void gethitched(long int val)
{
   printf("[***] Reindeer being hitched!!! ID: %ld \n", val);
   sleep(1);
}

void gethelp()
{
   sleep(1);
   if (elvesCount == 1)
       printf("\n[ - ] Elf waiting for help!!!\n");
   else
   {

       printf("[ - ] Elf waiting for help!!!\n");
   }
   sleep(1);
}
```

```
void helpElves()
{
   printf("[+++] Santa Helped Elves and toys are made!!!\n\n");
}

void *santa()
{
   while (1)
   {
       sem_wait(&santa_s);
       sem_wait(&mutex);
       if (reindeerCount == 9)
       {
           printf("\n[ * ] Santa Woke Up!!! ID: %ld\n", pthread_self());
           prepare();
           reindeerCount = 0;

           for (int j = 0; j < 9; j++)
               sem_post(&reindeer_s);
       }
       else if (elvesCount == 3)
       {
           sleep(1);
           printf("[---] Elves having difficulty to build toys!!!\n");
           printf("[ + ] Santa Woke Up!!! ID: %ld\n", pthread_self());
           helpElves();
       }
       sem_post(&mutex);
   }
}

void *reindeer()
{
   printf("[ - ] Reindeer Back from Vacation!!! ID: %ld\n",
pthread_self());
   while (1)
   {
       sem_wait(&mutex);
       reindeerCount += 1;
```

```
        if (reindeerCount == 9)
            sem_post(&santa_s);
        sem_post(&mutex);

        sem_wait(&reindeer_s);
        gethitched(pthread_self());
    }
}

void *elves()
{
    while (1)
    {
        sem_wait(&elf_s);
        sem_wait(&mutex);
        elvesCount += 1;
        int flag = 0;
        if (elvesCount == 3)
        {
            sem_post(&santa_s);
            flag = 1;
        }
        else

            sem_post(&elf_s);
        //if (flag == 1)
        //              printf("\n[ - ] Elves having difficulty to build
toys!!!\n");
        sem_post(&mutex);
        gethelp(elvesCount);

        sem_wait(&mutex);

        elvesCount -= 1;
        if (elvesCount == 0)
            sem_post(&elf_s);
        sem_post(&mutex);
    }
}
```

Paleti Krishnasai
CED18I039

```
int main()
{
   printf("\n");

   pthread_t san, elv[3], rend[9];
   sem_init(&santa_s, SHARED, 0);
   sem_init(&reindeer_s, SHARED, 0);
   sem_init(&elf_s, SHARED, 1);
   sem_init(&mutex, SHARED, 1);

   pthread_create(&san, NULL, santa, NULL);
   for (int i = 0; i < 9; i++)
       pthread_create(&rend[i], NULL, reindeer, NULL);
   for (int i = 0; i < 3; i++)
       pthread_create(&elv[i], NULL, elves, NULL);

   pthread_join(san, NULL);
   for (int i = 0; i < 3; i++)
       pthread_join(elv[i], NULL);
   for (int i = 0; i < 9; i++)
       pthread_join(rend[i], NULL);

   printf("\n");
   return 0;
}
```

Paleti Krishnasai
CED18I039

Output :

```
paleti@paletil:~/OS_LAB/Synchronization$ gcc santa.c -lpthread -lrt
paleti@paletil:~/OS_LAB/Synchronization$ ./a.out

[ - ] Reindeer Back from Vacation!!! ID: 140069672388352
[ - ] Reindeer Back from Vacation!!! ID: 140069655602944
[ - ] Reindeer Back from Vacation!!! ID: 140069663995648
[ - ] Reindeer Back from Vacation!!! ID: 140069647210240
[ - ] Reindeer Back from Vacation!!! ID: 140069638817536
[ - ] Reindeer Back from Vacation!!! ID: 140069630424832
[ - ] Reindeer Back from Vacation!!! ID: 140069622032128
[ - ] Reindeer Back from Vacation!!! ID: 140069613639424
[ - ] Reindeer Back from Vacation!!! ID: 140069605246720

[ * ] Santa Woke Up!!! ID: 140069680781056
[* *] Sleigh is being Prepared!!!
[***] Reindeer being hitched!!! ID: 140069663995648
[***] Reindeer being hitched!!! ID: 140069655602944
[***] Reindeer being hitched!!! ID: 140069613639424
[***] Reindeer being hitched!!! ID: 140069638817536
[***] Reindeer being hitched!!! ID: 140069630424832
[***] Reindeer being hitched!!! ID: 140069622032128
[***] Reindeer being hitched!!! ID: 140069605246720
[***] Reindeer being hitched!!! ID: 140069647210240
[***] Reindeer being hitched!!! ID: 140069672388352
[ - ] Elf waiting for help!!!
[---] Elves having difficulty to build toys!!!
[ + ] Santa Woke Up!!! ID: 140069680781056
[+++] Santa Helped Elves and toys are made!!!

[ - ] Elf waiting for help!!!
[ - ] Elf waiting for help!!!

[ * ] Santa Woke Up!!! ID: 140069680781056
[* *] Sleigh is being Prepared!!!
[***] Reindeer being hitched!!! ID: 140069655602944
[***] Reindeer being hitched!!! ID: 140069638817536
[***] Reindeer being hitched!!! ID: 140069672388352
[***] Reindeer being hitched!!! ID: 140069622032128
[***] Reindeer being hitched!!! ID: 140069630424832
[***] Reindeer being hitched!!! ID: 140069605246720
[***] Reindeer being hitched!!! ID: 140069647210240
[***] Reindeer being hitched!!! ID: 140069663995648
[***] Reindeer being hitched!!! ID: 140069613639424
```

Paleti Krishnasai
CED18I039

(2) H2O Problem

Code:

```
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#define SHARED 1

static int oxygen = 0, hydrogen = 0, count = 0, flag = 0;
sem_t mutex, hydro, oxy, barrier;

void bond()
{

   printf("[H2O] Molecules Bonded!!!\n");
   count += 1;

   if (count == 5)
   {
       printf("\n[ * ] Maximum Generated Water Molecules: %d\n\n", count);
       exit(1);
   }
   sleep(1);
}
void *hydrogen_thread()
{
   printf("[ H ] Hydrogen Molecule Generated: %ld\n", pthread_self());
   while (1)
   {
       sem_wait(&mutex);
       hydrogen += 1;
       if (hydrogen >= 2 && oxygen >= 1)
       {
           sem_post(&hydro);
           sem_post(&hydro);
           hydrogen -= 2;
           sem_post(&oxy);
           oxygen -= 1;
```

```
        }
        else
            sem_post(&mutex);

        sem_wait(&hydro);
        bond();

        sem_wait(&barrier);
    }
    pthread_exit(NULL);
}

void *oxygen_thread()
{
    printf("[ O ] Oxygen Molecule Generated: %ld\n", pthread_self());
    while (1)
    {
        sem_wait(&mutex);
        oxygen += 1;
        if (hydrogen >= 2)
        {
            sem_post(&hydro);
            sem_post(&hydro);
            hydrogen -= 2;
            sem_post(&oxy);
            oxygen -= 1;
        }
        else
            sem_post(&mutex);

        sem_wait(&oxy);
        bond();

        sem_wait(&barrier);
        sem_post(&mutex);
    }

    pthread_exit(NULL);
}
```

Paleti Krishnasai
CED18I039

```
int main()
{
   printf("\n");
   pthread_t hyd[10], ox[5];
   sem_init(&mutex, SHARED, 1);
   sem_init(&hydro, SHARED, 0);
   sem_init(&oxy, SHARED, 0);
   sem_init(&barrier, SHARED, 3);

   for (int i = 0; i < 10; i++)
       pthread_create(&hyd[i], NULL, hydrogen_thread, NULL);
   for (int i = 0; i < 5; i++)
       pthread_create(&ox[i], NULL, oxygen_thread, NULL);

   for (int i = 0; i < 10; i++)
       pthread_join(hyd[i], NULL);
   for (int i = 0; i < 5; i++)
       pthread_join(ox[i], NULL);

   printf("\n");
   return 0;
}
```

Paleti Krishnasai
CED18I039

Output:

```
paleti@paletil:~/OS_LAB/Synchronization$ ./a.out

[ H ] Hydrogen Molecule Generated: 139924012713728
[ H ] Hydrogen Molecule Generated: 139924004321024
[ H ] Hydrogen Molecule Generated: 139923995928320
[ H ] Hydrogen Molecule Generated: 139923987535616
[ H ] Hydrogen Molecule Generated: 139923979142912
[ H ] Hydrogen Molecule Generated: 139923970750208
[ H ] Hydrogen Molecule Generated: 139923962357504
[ H ] Hydrogen Molecule Generated: 139923953964800
[ H ] Hydrogen Molecule Generated: 139923937179392
[ O ] Oxygen Molecule Generated: 139923928786688
[ O ] Oxygen Molecule Generated: 139923920393984
[H2O] Molecules Bonded!!!
[ O ] Oxygen Molecule Generated: 139923912001280
[H2O] Molecules Bonded!!!
[H2O] Molecules Bonded!!!
[ H ] Hydrogen Molecule Generated: 139923945572096
[ O ] Oxygen Molecule Generated: 139923903608576
[ O ] Oxygen Molecule Generated: 139923895215872
[H2O] Molecules Bonded!!!
[H2O] Molecules Bonded!!!

[ * ] Maximum Generated Water Molecules: 5

[H2O] Molecules Bonded!!!
[H2O] Molecules Bonded!!!
paleti@paletil:~/OS_LAB/Synchronization$
```